PARALLEL IMPLICIT UNSTRUCTURED GRID EULER SOLVERS

V. Venkatakrishnan *
Institute for Computer Applications in Science and Engineering
MS 132C, NASA Langley Research Center
Hampton, VA 23681-0001

Abstract

A mesh-vertex finite volume scheme for solving the Euler equations on triangular unstructured meshes is implemented on an MIMD (multiple instruction/multiple data stream) parallel computer. An explicit four-stage Runge-Kutta scheme is used to solve two-dimensional flow problems. A family of implicit schemes is also developed to solve these problems, where the linear system that arises at each time step is solved by a preconditioned GMRES algorithm. Two partitioning strategies are employed, one that partitions triangles and the other that partitions vertices. The choice of the preconditioner in a distributed memory setting is discussed. All the methods are compared both in terms of elapsed times and convergence rates. It is shown that the implicit schemes offer adequate parallelism at the expense of minimal sequential overhead. The use of a global coarse grid to further minimize this overhead is also investigated. The schemes are implemented on a distributed memory parallel computer, the iPSC/860.

*Part of this work was done while the author was employed by Computer Sciences Corporation, Moffett Field, CA during which time the work was supported by the National Aeronautics and Space Administration under the NASA contract NAS2-12961. This research was also supported under the NASA contract No. NAS1-19480 while the author was in residence at the Institute for Computer Applications in Science and Enginnering (ICASE), NASA Langley Research Center, Hampton, VA 23681.

1 Introduction

Triangular meshes have become quite popular in computational fluid dynamics. They are ideal for handling complex geometries and for adapting to flow features, such as shocks and boundary layers. Considerable attention has been focused on improving the spatial operator, which has evolved to a very high degree of sophistication for the Euler and Navier-Stokes equations [1, 2]. Distributed memory parallel computers seem to offer an avenue for doing large problems very fast due to their scalability. For the goal of sustained high performance on these machines to be realized, many fundamental issues need to be addressed. Among these are scalable algorithms and software.

Explicit schemes used in computational fluid dynamics are completely parallel. They only require a simple update procedure that involves local dependencies. On a parallel computer, such schemes typically only require communication to nearest neighbors. Implicit schemes, on the other hand, require the solution of coupled equations which involves global dependencies. However, there are some applications, such as certain classes of unsteady flows, where explicit schemes are quite useful. When steady state solutions are sought, explicit schemes typically require thousands of time steps to converge and exhibit very slow convergence rates. Implicit schemes allow larger time steps to be taken and usually result in better convergence rates. Implicit schemes have to be designed carefully since the work involved at each time step could be substantial. Venkatakrishnan and Mavriplis [3] developed and tested a family of implicit schemes on the Cray Y-MP for solving the two-dimensional compressible Navier-Stokes equations on unstructured meshes. They concluded that the Generalized Minimal Residual technique of Saad and Schultz [4] with incomplete LU preconditioning (ILU(0)) based on the original nonzero pattern (GMRES/ILU) was by far the best implicit scheme over a whole range of flow conditions and was as efficient as the unstructured multigrid strategy of Mavriplis and Jameson [2]. On distributed-memory parallel computers, the design of implicit schemes is more difficult since parallelism and load balance during the implicit phase are additional considerations.

Venkatakrishnan et al. [5] and Das et al. [6] have shown that it is possible to obtain supercomputer performance when solving explicit unstructured grid problems on the iPSC/860. By paying careful attention to the partitioning of the mesh, communication schedule and data structures, they were able to show that 2-3 times the speed of a Cray Y-MP/1 could be obtained with 128 processors of the iPSC/860.

In this paper, two different ways of partitioning the unstructured mesh across processors are explored. The design of implicit schemes suitable for distributed-memory parallel computers is discussed next. The issues in implementing the GMRES algorithm and the preconditioners in parallel are addressed. Finally, results for a typical flow around a multi-element airfoil are presented and the performances of the explicit and implicit schemes on the iPSC/860 are compared using the two different partitioning strategies. The use of a coarse grid to improve convergence is also investigated with one of the implicit schemes.

2 Governing Equations and Spatial Discretization

The Euler equations in integral form for a control volume Ω with boundary $\partial\Omega$ read

$$\frac{d}{dt}\int_{\Omega} \boldsymbol{u}\, dv + \oint_{\partial\Omega} \boldsymbol{F}(\boldsymbol{u},\boldsymbol{n})\, dS = 0. \tag{1}$$

Here $\boldsymbol{u}$ is the solution vector comprised of the conservative variables density, the two components of momentum and total energy. The vector $\boldsymbol{F}(\boldsymbol{u},\boldsymbol{n})$ represents the inviscid flux vector for a surface with normal vector $\boldsymbol{n}$. The net efflux through the control volume boundary is termed the

residual. The variables $\boldsymbol{u}$ are stored at the vertices of a triangular mesh. The control volumes are nonoverlapping polygons which surround the vertices of the mesh. They form the *dual* of the mesh, which is composed of segments of medians. Associated with each edge of the original mesh is a (segmented) dual edge. The contour integrals in Eqn. (1) are replaced by discrete path integrals over these dual edges. The flux $\boldsymbol{F}(\boldsymbol{u}, \boldsymbol{n})$ is replaced by a numerical flux function. The construction of the numerical fluxes is done in two stages:

1. First, a piecewise linear reconstruction of the variables is performed. The variables are then interpolated to the edges of the control volumes. The gradients at the vertices which are required for this step are also computed as discrete path integrals over the edges of the control volmes.

2. The variables on either sides of the control volume edges are interpreted as initial data for Roe's approximate Riemann solver [7].

A Riemann problem is defined by Eqn. (1) subject to two constant states as initial data in one dimension. More details on the spatial discretization may be found in Barth and Jespersen [1]. A four-stage Runge-Kutta scheme is used to advance the solution in time for the explicit scheme. The residuals are evaluated by looping over the edges of the original mesh and vectorization is achieved by Barth and Jespersen by coloring the edges. For the parallel implementation, it is assumed that the triangulation of the computational domain and the mesh connectivity information are provided.

Two slightly different schemes are derived based on whether the cells or the vertices of the mesh are partitioned. For explicit schemes, identical solutions and convergence histories are obtained irrespective of this choice. The only difference lies in the way the communication is performed, which is discussed in the following paragraph. With the implicit schemes considered in this paper, on the other hand, one will in general obtain different convergence histories depending on this choice. The reason for this is that the schemes considered in this paper are only implicit within each processor (in the preconditioning phase). Cell partitioning leads to an overlap of vertices whereas vertex partitioning yields none and therefore, the two techniques produce different results. The treatment of the interpartition boundary vertices in the implicit scheme is a very important issue in the case of cell partitioning.

With cell partitioning, each triangle is assigned to a partition and the interpartition boundaries consist of edges of the original mesh. The vertices and the edges on the interpartition boundaries are duplicated. Figure 1 shows a triangular mesh under a 2-way cell partitioning. Also shown is the numbering of vertices local to each processor. Two communication phases are then required at each stage of the four-stage Runge-Kutta time integration, one during the computation of the gradients and the other, during the formation of the residuals. At an interpartition boundary vertex, which is shared by two or more partitions, each processor only sees a fraction of the control volume and thus computes only partial contributions to the integrals. The communication at the interpartition boundaries then consists of summing these local contributions to the integrals such as volumes, fluxes, and gradients. The communication phase is therefore termed the *combine* phase and involves floating point operations.

With vertex partitioning, each vertex is assigned uniquely to a partition and the interpartition boundaries consist of the dual edges (the edges of the control volumes). Figure 2 shows a triangular mesh under a 2-way vertex partitioning. The numbering of vertices is indicated as tuples, where the first index refers to the numbering for processor 0 and the second index refers to the numbering for processor 1. Again, two communication phases are required at each stage of the four-stage Runge-Kutta time integration. The communication required is different, however. The processors exchange data at two rows of vertices that are incident to the interpartition boundary edges. Now

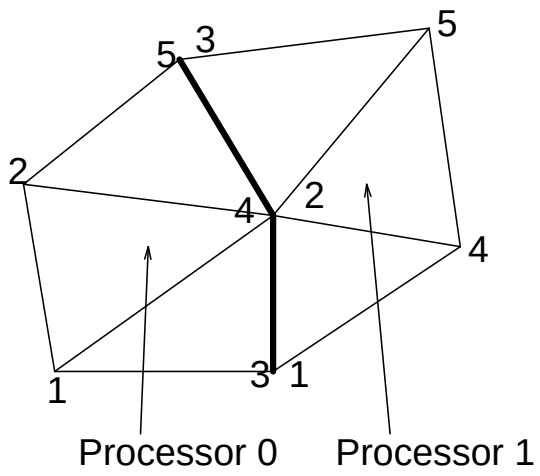

Figure 1: 2-way cell partitioning for a simple triangulation.

each processor can compute the entire integrals for all the vertices it owns. Thus, duplication of the flux calculations occurs at the interpartition boundary edges, but it is not a crucial issue on medium-grained parallel computers. Thus the communication phase with vertex partitioning also involves floating point operations.

Simon [8] has considered three different recursive partitioning strategies for partitioning unstructured meshes. In the context of an explicit two-dimensional Euler solver, Venkatakrishnan et al. [5] showed that the spectral bisection strategy was superior to the coordinate and graph bisection strategies. The spectral bisection technique produces uniform, mostly connected subdomains with short boundaries. Therefore, in this work, the recursive spectral bisection is used for partitioning.

After partitioning, global values of the data structures required to define the unstructured mesh are given local values within each partition. We thus dispense with any references to global indices. Das et al. [6] in their work develop primitives which allow access to global data address space, so that the transformation from global to local data structures is unnecessary. Access to global data structures is highly desirable for adaptive grid applications, but is not necessary when dealing with static meshes. In the present implementation, each local data set also contains the information that a partition requires for communication at its interpartition boundaries. The information required for communication at the interpartition boundaries is precomputed using sparse matrix data structures. Slightly different data structures are used depending on whether cell partitioning or vertex partitioning is employed. For details on these data structures, see [5]. Each subgrid is assigned to one processor. It was shown in [5] that the mapping of the subgrids to processors is not a crucial issue on the Intel iPSC/860. Therefore, a naive mapping that assigns subgrid 0 to processor 0, subgrid 1 to processor 1, and so on, is employed. Partitioning, conversion from global to local addresses, and generation of the data structures required for communication at the interpartition boundaries are all done on a workstation as a preprocessing step.

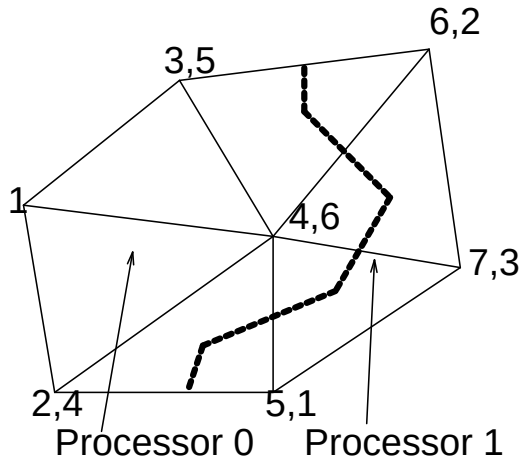

Figure 2: 2-way vertex partitioning for a simple triangulation.

3 Implicit scheme

After discretizing Eqn. (1) in space, the following system of coupled ordinary differential equations is obtained:

$$M\frac{dW}{dt} + R(W) = 0. \tag{2}$$

Here W is the vector of unknowns over all the mesh points. M is the mass matrix which represents the relationship between the average value in a control volume and the values at the vertices (the vertex representing the control volume and its nearest neighbors). It is only a function of the mesh and hence, a constant matrix for a static mesh. Since a steady state solution is sought, time-accuracy is not an issue and M can be replaced by the identity matrix yielding the following system of ordinary differential equations for the vector of unknowns W :

$$\frac{dW}{dt} + R(W) = 0. \tag{3}$$

If the time derivative is replaced by:

$$\frac{dW}{dt} = \frac{W^{n+1} - W^n}{\Delta t}, \tag{4}$$

an explicit scheme is obtained by evaluating $R(W)$ at time level n. An implicit scheme is obtained by evaluating $R(W)$ at level $n+1$. In the latter case, linearizing R about time level n, we obtain

$$\left(\frac{I}{\Delta t} + \frac{\partial R}{\partial W}\right)\Delta W_i = -R_i, \tag{5}$$

$$\Delta W_i = (W^{n+1} - W^n)_i. \tag{6}$$

Eqn. (5) represents a large nonsymmetric linear system of equations for the updates of the vector of unknowns and needs to be solved at each time step. As Δt tends to infinity, the method reduces to the standard Newton's method. The term $\frac{\partial R}{\partial W}$ symbolically represents the implicit

side upon linearization and involves the Jacobian matrices of the flux vectors with respect to the conservative variables. Due to storage considerations, only a lower order representation of the operator is employed on the implicit side. Thus the graph of the sparse matrix $\frac{\partial R}{\partial W}$ is identical to the graph of the supporting unstructured mesh (i.e., each vertex is only connected to its nearest neighbors). The sparse matrix thus has a symmetric structure, even though the matrix is itself unsymmetric in general. A complete linearization of $R(W)$ will result in a much denser matrix with a different graph, since each vertex will also be connected to its next-to-nearest neighbors. The storage requirements for such a scheme are prohibitive even for two-dimensional meshes. The penalty in making this approximation in the linearization is that Eqn. (5) can never approach Newton's method (with its associated quadratic convergence property) due to the mismatch of the right and left hand side operators.

Since the linear system is itself approximate, there is little to be gained by solving it to a great precision. To obtain favorable overall (nonlinear) convergence, it has been found that it is better to solve the linear problem to a moderate degree of precision and proceed to the next time step. However, for stiff problems it may well be necessary to solve the linear problem to high precision and one has the control to do so in the present framework. The time step in Eqn. (5) is allowed to vary inversely proportional to the L_2 norm of the residual. Since there is a mismatch of operators in Eqn. (5), it is also necessary to limit the maximum time step.

There is a host of methods in linear algebra literature for solving nonsymmetric systems of linear equations, but in this work only the GMRES technique developed by Saad and Schultz [4] is considered. The GMRES technique is quite efficient for solving sparse nonsymmetric linear systems and is outlined below. Let x_0 be an approximate solution of the system

$$Ax + b = 0 \tag{7}$$

where A is an invertible matrix. The solution is advanced from x_0 to x_k as

$$x_k = x_0 + y_k \tag{8}$$

GMRES(k) finds the best possible solution for y_k over the Krylov subspace $< v_1, Av_1, A^2v_1, ..., A^{k-1}v_1 >$ by solving the minimization problem

$$||r_k|| = Min_y||v_1 + Ay|| \tag{9}$$

$$v_1 = Ax_0 + b, r_k = Ax_k + b \tag{10}$$

GMRES procedure forms an orthogonal basis $v_1, v_2, ...v_k$ (termed search directions) spanning the Krylov subspace by a modified Gram-Schmidt method. These search directions need to be stored. As k increases, the storage increases linearly and the number of operations, quadratically. To mitigate this, Saad and Schultz also describe GMRES (k, m) which is a restarted GMRES (k), where the k search directions are discarded and recomputed every m cycles. GMRES can also be thought of as an optimal polynomial acceleration scheme. Preconditioning greatly improves the performance of GMRES as well as the other related iterative methods. It clusters the eigenvalues around unity so that the optimal polynomial generated by GMRES can better annihilate the errors associated with each eigenvalue.

4 Preconditioning and parallelism issues

Instead of Eqn. (7) the preconditioned iterative methods solve the following system:

$$AQ(Q^{-1}x) + b = 0 \tag{11}$$

The system of linear equations in Eqn. (11) is referred to as the right preconditioned system and Q as the right preconditioner. Right preconditioning is preferred over left preconditioning for reasons given in [3]. The role of the preconditioner is to cluster the eigenvalues around unity. The choice of preconditioners and the issues in implementing the preconditioned GMRES on the parallel computer are addressed below.

On a distributed-memory parallel computer, the same least squares problem of Eqn. (9) is seen by all the processors. While this results in some duplication of work, the main nonlocal kernels of the GMRES are distributed across multiple processors. These kernels include sparse matrix - vector multiplication, dot products and L_2 norm evaluations. Whereas on a Cray Y-MP, vectorization for the sparse matrix-vector product is achieved by using an edge-oriented data structure for the matrix and coloring the edges of the graph, this is not the optimal way to compute the matrix vector product on a parallel computer, where locality is of utmost importance. This is true even when dealing with a single node of the parallel computer. Therefore the usual row oriented sparse matrix data structure is used. We have found that even on a single node this approach outperforms the one that uses the edge-based data structure by a factor of two. In the case of cell partitioning, the rows corresponding to the interpartition boundary vertices are distributed across the processors sharing them while the remaining rows are assigned uniquely to processors. In the case of vertex partitioning, the rows are uniquely assigned to processors. Akin to the explicit scheme, each processor computes its share of the matrix vector multiplication. The communication step with cell partitioning consists of summing the local contributions at the interpartition boundaries. The communication step with vertex partitioning consists of exchange of the vector components at the two rows of vertices incident to the interpartition boundary edges. This is followed by the matrix - vector multiplication at each processor. More details on the implementations of the matrix - vector product may be found in [9].

In most practical problems of interest, the choice of the preconditioner is very important but the effort involved in applying the preconditioner should not be prohibitive. The implicit scheme without preconditioning possesses complete parallelism, except for the duplication of some work when solving the least squares problem in GMRES. On a parallel computer, the parallelism in the preconditioning phase is an important additional consideration. A simple choice is a block diagonal preconditioner which computes the inverse of the 4x4 diagonal block associated with a mesh point. The LU decomposition of the 4×4 blocks and the forward and back solves are local and hence, are inherently parallel. In [3] an ILU(0) preconditioner was also considered. ILU(0) refers to an incomplete factorization with no fill-in beyond the original non-zero pattern. By using a level scheduling [10] (also known as wavefront ordering) it is possible to obtain parallelism with this preconditioner. Under this permutation of the matrix, unknowns within a wavefront can be eliminated simultaneously. However, since the degree of parallelism varies with the wavefront, it cannot be easily exploited on a distributed-memory parallel computer. A fixed partitioning strategy for the mesh incurs substantial load imbalance, while a dynamic partitioning strategy entails substantial data movement and hence, increased communication costs. Barszcz et al. [11] have found that using a fixed partitioning strategy when solving triangular systems of equations on a regular grid results in low upper bounds on efficiency even in the absence of communication. Therefore, for general sparse matrices, the ILU(0) preconditioner is ill-suited for implementation on a distributed-memory parallel computer. Therefore, we settle on an ILU preconditioner that is processor-implicit i.e., ILU(0) is carried out for all the vertices internal to a processor. Thus, at a macro-level, the overall preconditioner can be viewed as an approximate block Jacobi iteration, wherein each block is assigned to a processor and an approximate LU factorization viz. ILU(0) is carried out. A block here refers to a macro-block consisting of all the unknowns assigned to a processor. When cell partitioning is employed, a block diagonal preconditioner is used for all the

vertices on the inter-processor boundaries, where a block refers to the 4×4 matrix associated with each vertex. The overall preconditioner is weaker than the global ILU(0), and degenerates to a block diagonal preconditioner in the limit of one grid point per processor. Thus, as the number of processors increases, degradation in convergence is to be expected. This degradation should be moderate, since the iPSC/860 is coarse-grained parallel computer.

In order to improve the preconditioner at the interpartition boundary vertices when using cell partitioning, domain decomposition techniques have also been examined. To this end, the modified Schur complement preconditioning (MSC) of Keyes and Gropp [12] with a block diagonal approximation for the Schur complement is implemented. The entries of this block diagonal approximation are determined by the probing technique of Chan and Resasco [13] and Keyes and Gropp [12]. Since a 4×4 block diagonal approximation is sought, the probe vectors used are e_1, e_2, e_3 and e_4, where e_j is the jth unit column vector. The probing technique allows for a banded approximation to the Schur complement. However, banded approximations for the Schur complement pose problems in a parallel setting. It is also not clear as to how do deal with cross points (vertices that are shared by more than two processors). Therefore, only a block diagonal approximation is investigated.

When using vertex partitioning, there is no overlap of vertices between processors, and hence no special interface preconditioning is required when GMRES/ILU is used. To facilitate communication, vertices on either sides of the partition boundaries are duplicated, thus leading to a minimal overlap. In the preconditioning phase, ILU factorization is carried out for each processor by zeroing out the matrix entries whose column numbers lie outside the processor domain. This is equivalent to solving the problem within each processor with zero Dirichlet boundary conditions during the preconditioning. This approximation is consistent with the steady state solution $\Delta W \equiv 0$ everywhere. Again, degradation in convergence is to be expected as the number of processors increases. More iterations will be required to obtain the same level of convergence. This additional work will be called the sequential overhead.

In order to minimize the sequential overhead, we appeal to techniques developed in domain decomposition. One of the most successful methods in use in domain decomposition is the Schwartz alternating procedure, which can also be implemented as a preconditioner. The scheme outlined above is an example of an additive Schwartz preconditioner. The term additive denotes that the preconditioner can be carried out in parallel. This is in contrast to the multiplicative Schwartz method, which requires that the preconditioner be applied in a sequential way by cycling through the processors in some order. It is possible to extract some coarse-grained parallelism by coloring the subdomains, but the potential is limited. Therefore in a parallel context, the additive Schwartz method is preferred.

A powerful idea for elliptic problems advocated by Dryja and Widlund [15], is the use of a coarse grid in order to bring some global influence to bear on the problem, similar in spirit to a two-level multigrid algorithm. The coarse grid operator is applied multiplicatively i.e., it is solved first and its solution is used during the additive phase. Applying the coarse grid in this manner does impose a penalty in a parallel setting; it becomes a sequential bottleneck. Cai et al. [16] have applied the multiplicative and additive Schwartz algorithms to the solution of symmetric and nonsymmetric problems. They have observed almost h-independent convergence, where h is the fine grid size, provided the coarse grid is fine enough. In some instances, the coarse grid operator may be formed by discretizing the partial differential equation on a coarse grid. However, in our application, this will require a triangulation followed by a discretization on this coarse grid. Generation of interpolation operators to transfer information between the coarse and the fine grids is also necessary. We avoid all these complexities by appealing to an alternative way of obtaining a coarse grid operator described by Wesseling [17]. Using this method, a coarse grid Galerkin operator is easily derived from a given fine grid operator, restriction and prolongation operators. We choose

the restriction operator to be a simple summation of fine grid values, and the prolongation operator to be injection. Under this choice, the coarse grid discretization can be shown to be equivalent to be similar to the one used in an agglomeration multigrid strategy, details of which are given in a companion paper [18]. It amounts to identifying all the vertices that belong to a processor by one coarse grid vertex, and summing the equations and the right hand sides associated with them. Thus the coarse grid system has as many vertices as the number of subdomains. At each time step, a coarse grid system is formed and solved by using a direct solver. The data obtained from the coarse grid is used on the boundaries as Dirichlet data for each processor. We have found that in practice, a direct solver is seldom needed to solve the coarse grid system; an iteration of incomplete LU decomposition seems to suffice. We have also found that at least one cycle of implicit smoothing similar to that employed in multigrid context by Mavriplis and Jameson [2] is needed to mitigate the adverse effects of injection of the solution from the coarse to the fine grid. On the fine grid, after injection, one Jacobi iteration is performed on the following system of implicit equations:

$$(I + \epsilon d) u_i^{new} = u_i^{old} + \sum_{j=1}^{d} \epsilon u_j^{new}, \tag{12}$$

where ϵ is taken to be 0.5, d is the degree of the vertex i, and the summation is over the neighbors of each vertex. This smoothing step involves communication at the boundaries.

The vertex-partitioned scheme is more amenable to the investigation of the coarse grid operator than the cell-partitioned scheme, and therefore, we only discuss this particular implementation. Each processor first forms parts of the coarse grid matrix and the right hand side at every time step. A global concatenation is performed so that each processor has the entire coarse grid system. This system is solved redundantly by each processor by forming approximate L and U factors. During the preconditioning phase, each processor forms a portion of the right hand side. After a global concatenation, each processor carries out forward and backward solves and deduces the appropriate Dirichlet data. We have also developed a weaker smoother that dispenses with communication associated with the Jacobi smoothing, but yields comparable convergence. This technique termed *modified Jacobi smoothing*, smooths the neighboring coarse grid data (to be used as Dirichlet data) with the data that the processor holds. This step is given by the following relation:

$$(I + \epsilon) U_D^{new} = U_D + \epsilon U_{LOC}, \tag{13}$$

where U_D is the old Dirichlet data, U_D^{new} is the new Dirichlet data and U_{LOC} is the value of the coarse grid vertex assigned to the processor.

5 Performance on the Intel iPSC/860

Subcritical flow past a four-element airfoil in a landing configuration with a freestream Mach number $M_\infty = 0.2$ and an angle of attack of 5° is considered as a test case. Supercritical flows are not considered here because convergence is a problematic issue whenever limiters are employed and therefore, it is difficult to study the effect on convergence as the number of processors increases. However, if the fixes to limiters, as proposed in [14], are implemented, the trends are expected to be similar to those for subcritical flows.

Performance results are presented for two problem sizes that are representative of two-dimensional inviscid flows. The coarse mesh has 6019 vertices, 17,473 edges, 11,451 triangles, 4 bodies, and 593 boundary edges. The fine mesh has 15,606 vertices, 45,878 edges, 30,269 triangles, 4 bodies, and 949 boundary edges. In the Cray implementation of the explicit code vectorization is achieved

by coloring the edges of the mesh (more details may be found in [1]. The Cray implementation is highly optimized and the performance of the code is comparable to that of existing structured grid codes.The implicit code was not optimized for the Cray Y-MP, since it was developed on the Intel iPSC/860. The result is that it runs in an almost scalar fashion on the Cray, except for the right hand side computation. However, a similar implicit unstructured mesh Navier-Stokes code was implemented earlier on the Cray Y-MP and optimized by Venkatakrishnan and Mavriplis [3], which runs at around 110–120 megaflops. All the megaflop numbers in this section are based on operation counts using the Cray hardware performance monitor.

The explicit scheme is a four stage Runge-Kutta scheme and uses a CFL number of 1.4. With the GMRES/DIAG scheme, the start-up CFL number is 3 and the CFL number is allowed to vary inversely proportional to the L_2 norm of the residual up to a maximum of 30. With GMRES/ILU, the start-up CFL number is 20 and the CFL number is allowed to vary inversely proportional to the L_2 norm of the residual up to a maximum of 200000. With both the implicit schemes, the number of GMRES search directions used is 15.

First, the explicit and implicit schemes are compared when cell partitioning is employed. When using GMRES/ILU, a block diagonal preconditioning is employed for the vertices on the interprocessor boundaries. Figures 3 and 4 show the convergence characteristics of the explicit and implicit schemes as a function of the number of iterations for the coarse and fine meshes.

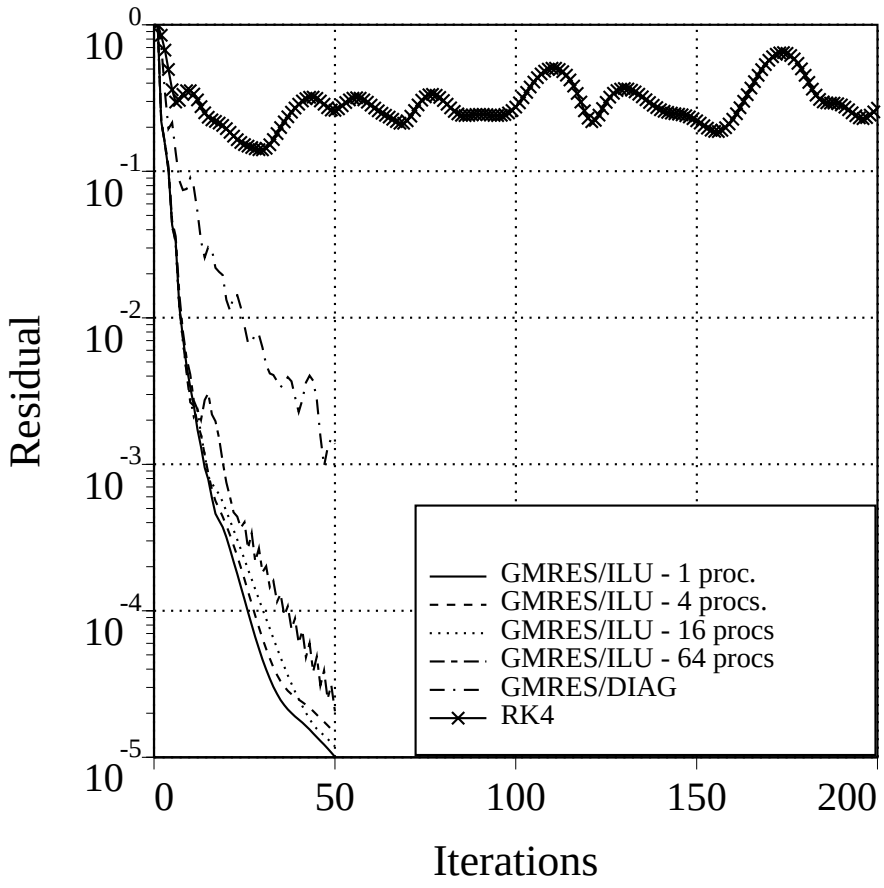

Figure 3: Convergence histories with various schemes on the coarse mesh with cell partitioning.

It may be observed that the explicit scheme is barely converging while the implicit schemes converge quickly. The GMRES/ILU processor-implicit preconditioning exhibits degradation in

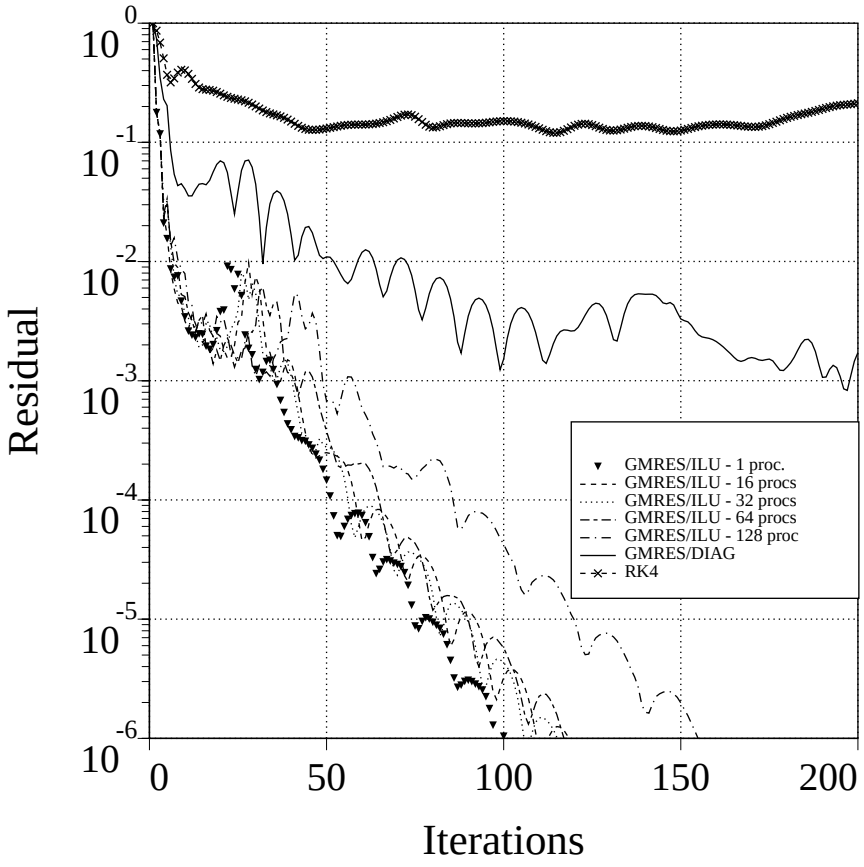

Figure 4: Convergence histories with various schemes on the fine mesh with cell partitioning.

convergence as the number of processors increases, but the degradation is moderate. Even with 128 processors, it performs much better than the GMRES/DIAG. In examining Figures 3 and 4, it is seen that the convergence histories with GMRES/ILU gravitate towards that of GMRES/DIAG as the number of processors increases. In the limit of 1 grid point per processor the two will be identical. In these figures, since the problem does not fit on one processor of the Intel iPSC/860, the uni-processor runs were carried out on the Cray Y-MP. The times per iteration in seconds and the convergence rates are shown in Tables 1 and 2 for the coarse and fine meshes, respectively. The convergence rate is defined as

$$Rate = \left(\frac{R_n}{R_1}\right)^{\frac{1}{n-1}} \tag{14}$$

where R_n is the L_2 norm of the residual at the end of nth time step and R_1 is the residual at the end of the first time step. It may be observed that the convergence rates of the explicit scheme (RK4) and the implicit scheme (GMRES/DIAG) are independent of the number of processors used, whereas that of GMRES/ILU exhibits a slight degradation with increasing number of processors. Finally, since time to completion is of ultimate interest, Figure 5 shows the convergence histories for the fine mesh problem as a function of the elapsed time with the number of processors fixed at 64. It clearly shows the superiority of the GMRES/ILU processor-implicit technique over the explicit and the GMRES/DIAG schemes.

The effect of improving the preconditioning of the interface vertices by means of the Modified Schur Complement preconditioning technique [12] is addressed next. A very coarse grid about an

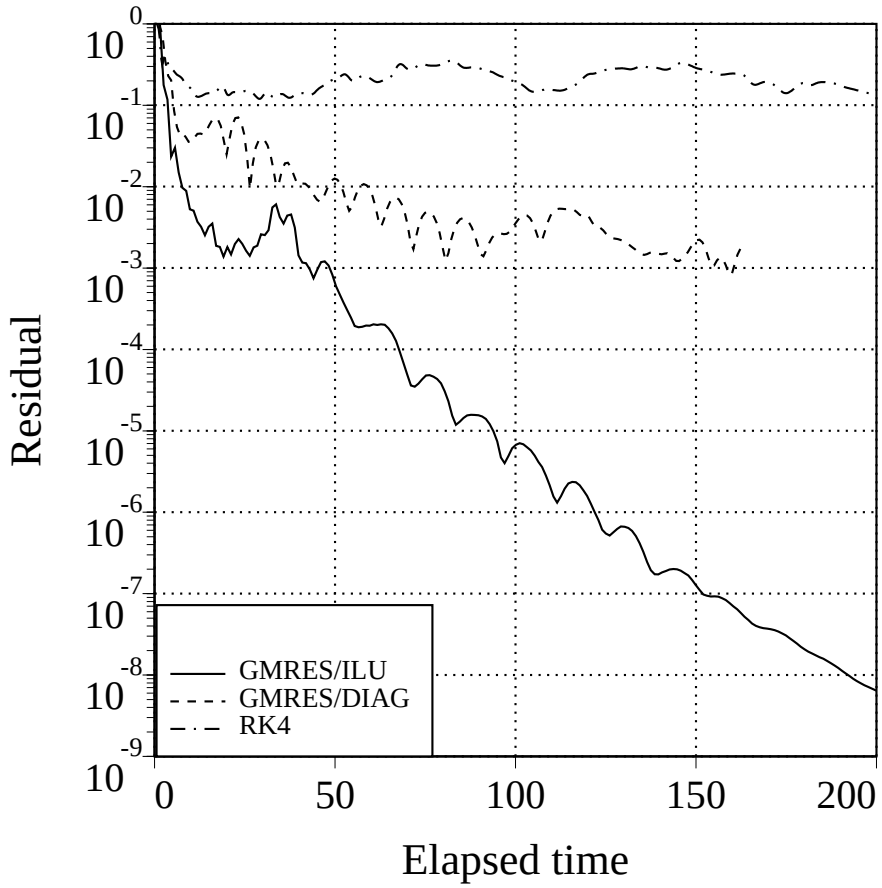

Figure 5: Convergence histories as a function of elapsed times on the fine mesh with 64 processors.

NACA0012 airfoil with 460 vertices is considered under subsonic flow conditions. For simplicity, only a two processor partitioning is considered. In this case, the vertices on the interface are only shared by two processors (i.e., there are no cross points). The theory of MSC preconditioning is based on considering a domain split into two nonoverlapping domains with vertices on the interface. In this work a block diagonal approximation to the Schur complement is derived by using the probe vectors discussed earlier. Figure 6 shows the convergence histories as a function of the number of iterations of the GMRES/ILU technique with block diagonal preconditioning and with MSC preconditioning. The convergence deteriorates with MSC preconditioning compared to using diagonal preconditioning. In light of this negative result and of the heuristic nature by which MSC preconditioning is extended to deal with situations involving more than two processors and having cross points, we do not investigate this technique any further. We believe the block diagonal preconditioning for the interface vertices offers the best compromise in terms of parallelism and convergence.

Next, the explicit and implicit schemes are compared when vertex partitioning is employed. Figures 7 and 8 show the convergence histories for the coarse and fine meshes when using vertex partitioning. The convergence histories are comparable to those obtained with cell partitioning (Figures 3 and 4). Tables 3 and 4 show the times per iteration in seconds and the convergence rates. In comparing with the results for cell partitioning (Tables 1 and 2), it is seen that the elapsed times with vertex partitioning are slightly better than those with cell partitioning for all the algorithms, especially for the implicit schemes. In addition, as the number of processors increases,

Table 1: **Performance of the cell-partitioned implicit scheme on the Intel iPSC/860 - 6019 vertices**

Scheme	Measure	No. of processors					
		1	4	8	16	32	64
RK4	Time/iteration (sec)	-	1.06	0.58	0.33	0.20	0.13
	Conv. rate/iteration	0.973	0.973	0.973	0.973	0.973	0.973
GMRES/DIAG	Time/iteration (sec)	-	3.12	1.72	1.00	0.64	0.46
	Conv. rate/iteration	0.874	0.874	0.874	0.874	0.874	0.874
GMRES/ILU	Time/iteration (sec)	-	4.47	2.40	1.34	0.82	0.55
	Conv. rate/iteration	0.791	0.797	0.798	0.793	0.799	0.802

Table 2: **Performance of the cell-partitioned implicit scheme on the Intel iPSC/860 - 15606 vertices**

Scheme	Measure	No. of processors				
		1	16	32	64	128
RK4	Time/iteration (sec)	-	0.85	0.44	0.25	0.16
	Conv. rate/iteration	0.997	0.997	0.997	0.997	0.997
GMRES/DIAG	Time/iteration (sec)	-	2.27	1.30	0.81	0.56
	Conv. rate/iteration	0.968	0.968	0.968	0.968	0.968
GMRES/ILU	Time/iteration (sec)	-	3.16	1.74	1.04	0.68
	Conv. rate/iteration	0.870	0.880	0.883	0.885	0.903

the vertex-partitioned GMRES/ILU scheme exhibits less degradation in convergence in comparison with the cell-partitioned scheme. The convergence rates shown in Tables 3 and 4 bear this out as well. The vertex-partitioned GMRES/ILU scheme requires only about 20% more iterations with 128 processors than the ideal 1 processor scheme to obtain the same level of convergence (5 orders of reduction in the residual norm).

Finally, we examine the effects of using a coarse grid as discussed in the last section to improve convergence for the 15,606 vertices mesh. Figure 9 shows the convergence histories as a function of iterations for the uni-processor, 32-processor and 128-processor cases with and without the use of a coarse grid. A cycle of modified Jacobi smoothing is employed as part of the preconditioner in order to stabilize the procedure with the coarse grid system and, the coarse grid system is solved redundantly by all processors. The convergence has improved dramatically, illustrating the power of the coarse grid. The convergence with 128 processors is even better than that obtained with the uni-processor scheme. Unfortunately, this improved convergence does not translate into a reduction in the time required to solve the problem. This is illustrated in Figure 10, which shows the convergence histories as a function of elapsed times on 32 and 128 processors with and without the coarse grid. In both the 32 and the 128-processor cases, it may be observed that the times required to solve the problem are nearly the same with and without the use of the coarse grid. On a per iteration basis, the elapsed times for the 32-processor case are 1.73 and 1.87 seconds respectively, without and with the coarse grid. For the 128-processor case, these times are 0.64 an 1.02 seconds. This points to a major drawback of using the coarse grid system to improve convergence. With too small a coarse grid system, the effort required to solve the system is minimal, but so is the improvement in

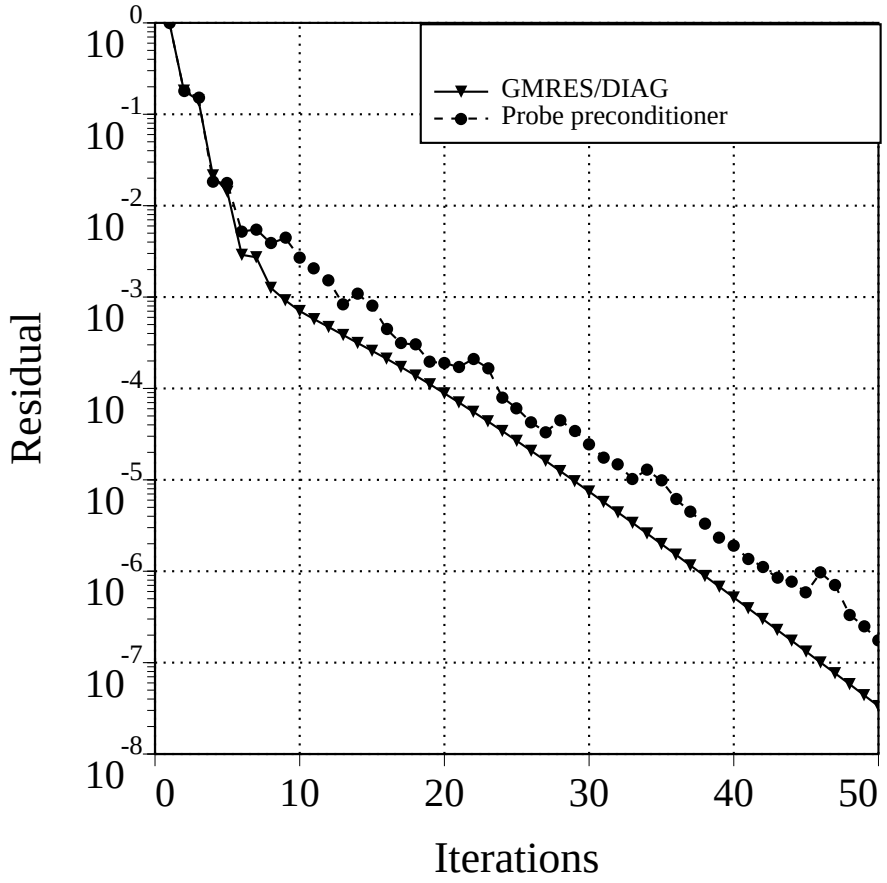

Figure 6: Convergence histories with diagonal preconditioning and with MSC preconditioning on a coarse mesh with 460 vertices.

convergence. With a larger coarse grid system, the gain in convergence is substantial but comes at a greater cost. The coarse grid operator, being sequential in nature, predominates as the number of processors increases.

In order to get an idea of the relative performances of the codes on the Intel iPSC/860 and the Cray Y-MP/1, performance data from the Cray implementation are given. The elapsed times on the Cray Y-MP/1 are respectively, 0.15 and 0.39 seconds per time step for the coarse and fine meshes with the explicit scheme. The explicit code runs at 150 megaflops on the Cray Y-MP/1. The megaflop ratings on the Cray are obtained using the hardware performance monitor. Timings for the implicit scheme on the Cray Y-MP/1 are not provided since the codes have not been not optimized.

6 Conclusions

It has been shown that implicit schemes can be carefully designed to yield good performance when solving unstructured grid problems on parallel computers. Such schemes outperform explicit schemes in terms of the time required to converge to a steady state solution. The GMRES technique with ILU preconditioning within each processor performs better than the explicit scheme and the GMRES procedure with block diagonal preconditioning. In the case where cells are partitioned, a block diagonal preconditioning for the interface points is shown to be effective. The degradation in

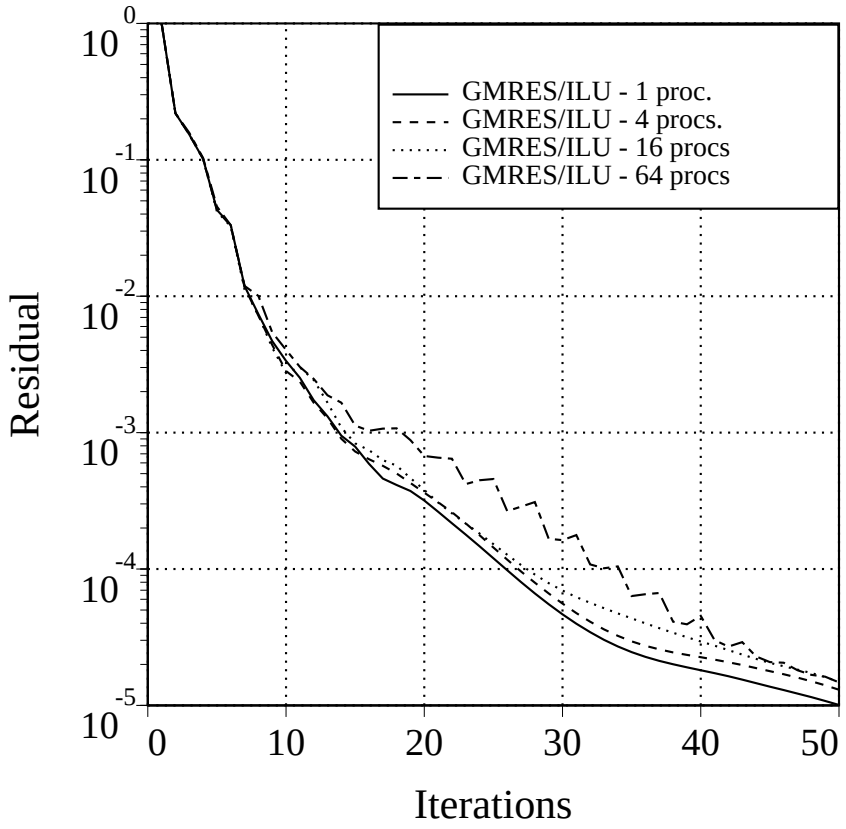

Figure 7: Convergence histories with GMRES/ILU on the coarse mesh with vertex partitioning.

convergence with increasing number of processors that results from this treatment of the interface nodes is shown to be moderate. It is shown that vertex-partitioning yields better results, both in terms of elapsed times and convergence rates. The implicit schemes presented in this paper offer almost complete parallelism at the expense of minimal sequential overhead, thus resulting in efficient parallel algorithms. Finally, it is demonstrated that the use of a global coarse grid, while improving convergence, does not result in a reduction in the time to solve the problem on the Intel iPSC/860.

7 Acknowledgements

The author thanks the Numerical Aerodynamics Simulation facility at NASA Ames Research center for the use of the Intel iPSC/860. The author also thanks Tim Barth of the Computational Fluid Dynamics branch for providing the explicit Cray code, and David Keyes and Tidiri Driss of ICASE for the many helpful discussions on domain decomposition methods.

References

[1] Barth, T.J., and Jespersen, D., "The design and application of upwind schemes on unstructured

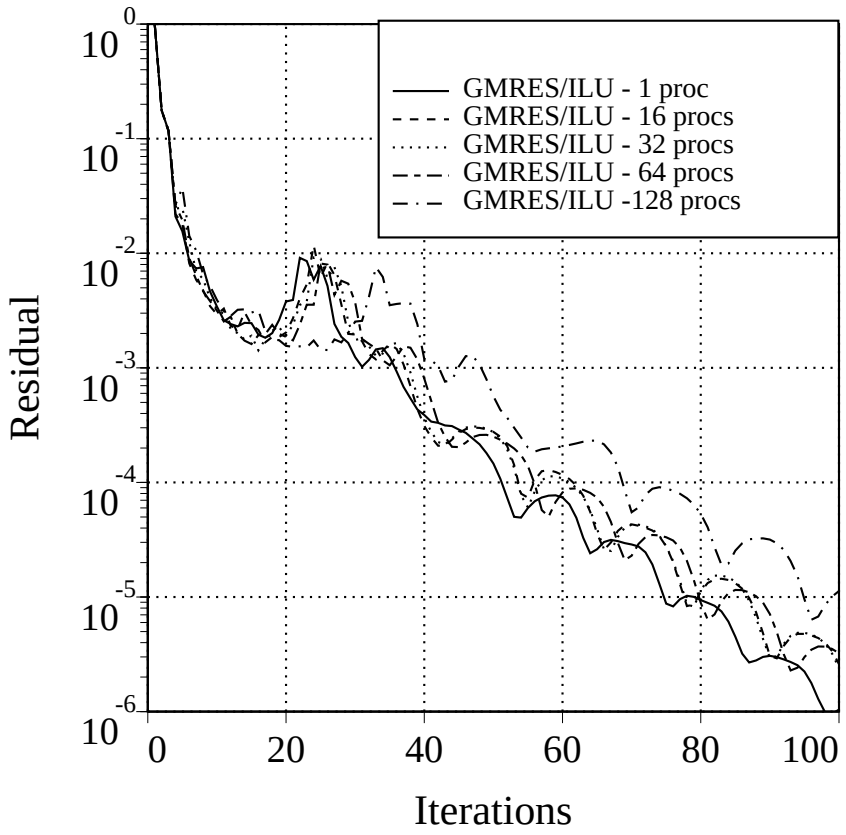

Figure 8: Convergence histories with GMRES/ILU on the fine mesh with vertex partitioning.

meshes. In *27th Aerospace sciences Meeting* AIAA 89-0366, Reno, NV, 1989.

[2] Mavriplis, D.J., and Jameson, A., "Multigrid solution of the two-dimensional Euler equations on unstructured triangular grids", AIAA Journal, Vol 26, No. 7, July 1988, pp. 824-831, 1988.

[3] Venkatakrishnan, V., and Mavriplis, D.J., "Implicit solvers for unstructured meshes", *J. Comp. Physics*, Vol. 105, pp. 83–91, 1993.

[4] Saad, Y., and Schulz, M.H., "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems", *SIAM J. Sci. Stat. Comp.*, Vol. 7, No. 3, pp. 856–869, 1986.

[5] Venkatakrishnan, V., Simon, H.D., and Barth, T.J., "A MIMD Implementation of a Parallel Euler Solver for Unstructured Grids", *The Journal of Supercomputing*, 6, pp. 117–137, 1992.

[6] Das, R., Mavriplis, D.J., Saltz,J., Gupta,S., and Ponnusamy, R., "The design and implementation of a parallel unstructured Euler solver using software primitives", in *30th AIAA Aerospace Sciences Conf.*, AIAA 92-0562, Reno, NV, 1992.

[7] Roe, P. L., "Characteristic-based Schemes for the Euler Equations", *Annual Review of Fluid Mechanics,* Vol. 18, 1986, pp. 337–365.

[8] Simon, H.D., "Partitioning of unstructured problems for parallel processing", *Computing Systems in Engineering*, Vol. 2, No. 2/3, pp. 135-148, 1991.

Table 3: **Performance of the vertex-partitioned implicit scheme on the Intel iPSC/860 - 6019 vertices**

Scheme	Measure	No. of processors					
		1	4	8	16	32	64
RK4	Time/iteration (sec)	-	1.07	0.59	0.32	0.20	0.13
	Conv. rate/iteration	0.973	0.973	0.973	0.973	0.973	0.973
GMRES/DIAG	Time/iteration (sec)	-	3.06	1.66	0.95	0.59	0.42
	Conv. rate/iteration	0.874	0.874	0.874	0.874	0.874	0.874
GMRES/ILU	Time/iteration (sec)	-	4.42	2.36	1.32	0.77	0.52
	Conv. rate/iteration	0.791	0.795	0.796	0.797	0.797	0.797

Table 4: **Performance of the vertex-partitioned implicit scheme on the Intel iPSC/860 - 15606 vertices**

Scheme	Measure	No. of processors				
		1	16	32	64	128
RK4	Time/iteration (sec)	-	0.78	0.43	0.25	0.15
	Conv. rate/iteration	0.997	0.997	0.997	0.997	0.997
GMRES/DIAG	Time/iteration (sec)	-	2.19	1.24	0.75	0.51
	Conv. rate/iteration	0.968	0.968	0.968	0.968	0.968
GMRES/ILU	Time/iteration (sec)	-	3.07	1.73	1.07	0.65
	Conv. rate/iteration	0.870	0.878	0.878	0.880	0.891

[9] Venkatakrishnan, V., "Parallel computation of Ax and $A^T x$", To appear in *Int. J. of High Speed Computing*, 1994.

[10] Anderson, E., and Saad, Y., "Solving sparse triangular systems on parallel computers", *International J. of High Speed Computing*, Vol. 1, No. 1, pp. 73–96, 1989.

[11] Barszcz, E., Fatoohi,R.A., Venkatakrishnan, V., and Weeratunga,S., "Solution of regular, sparse triangular linear systems on vector and distributed-memory multiprocessors", NAS Applied Research Branch technical report RNR-93-007, NASA Ames Research Center, Moffett Field, CA, 1993.

[12] Keyes, D.E., and Gropp, W.D., "Domain decomposition techniques for nonsymmetric systems of elliptic boundary value problems: Examples from CFD.", in the Proceedings of *The Second International Symposium on Domain Decomposition Methods*, UCLA, CA, 1988.

[13] Chan, T.F., and Resasco, D.C., "A framework for the analysis and construction of domain decomposition preconditioners", *UCLA Dept. of Computational and Applied Mathematics Report*, CAM 87-09, 1987.

[14] Venkatakrishnan, V., "On the accuracy of limiters and convergence to steady state solutions", Presented at *31st AIAA Aerospace Sciences Meeting*, AIAA 93-0880, Reno, NV, 1993. Submitted to the J. of Comp. Phys.

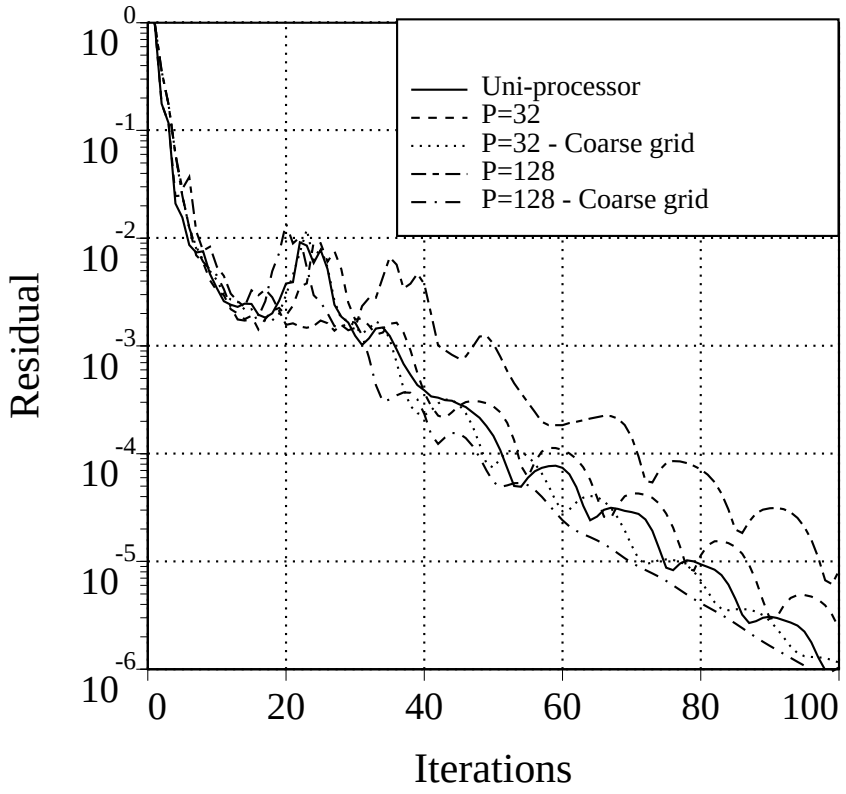

Figure 9: Convergence histories for the 15606 case as a function of iterations with and without the use of a coarse grid.

[15] Dryja, M., and Widlund, O.B., "Towards a Unified Theory of Domain Decomposition Algorithms for Elliptic Problems", in *Third Intl. Symp. on Domain Decomposition Methods for Partial Diff. Eqns.*, T.Chan, R. Glowinski, J.Periaux, and O. Widlund, eds., SIAM, Philadelphia, 1990, pp. 3–21.

[16] Cai, X., Gropp, W.D., and Keyes, D.E., "A Comparison of Some Domain Decomposition and ILU Preconditioned Algorithms for Nonsymmetric Elliptic Problems", to appear in *J. Numer. Linear Alg. with Appl.*, 1994.

[17] Wesseling, P., "An Introduction to Multigrid Methods", *John Wiley & Sons*, 1992.

[18] Venkatakrishnan, V., and Mavriplis, D.J., "Agglomeration Multigrid for the Three-dimensional Euler Equations", *AIAA Paper 94-0069*, January 1994.

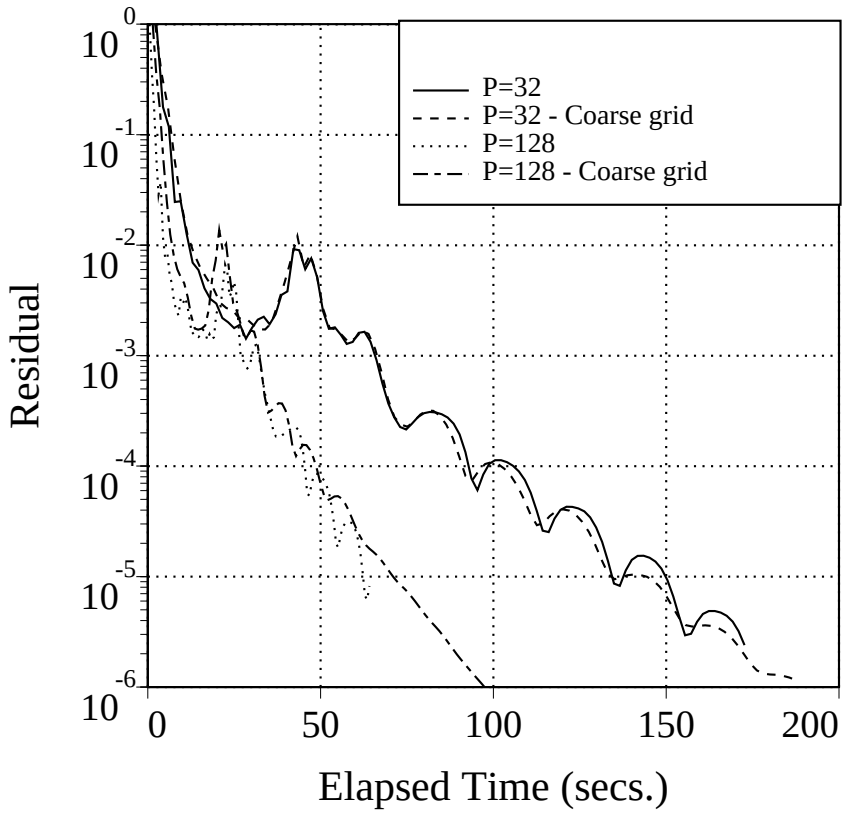

Figure 10: Convergence histories 15606 vertices case as a function of elapsed times with and without the use of a coarse grid.